Vol.2011-SE-171 No.17
2011/3/15

日本語で一貫したプログラミングの実践 ～プロデルを用いて～

馬場 祐人[†] 筧 捷彦[††]

現在の日本におけるソフトウェア開発では母国語である日本語で仕様書を作成し，Java に代表されるような欧米発想のプログラミング言語を使って実装している．本来日本語と欧米言語とは表現や意味にギャップがあり，そのことで仕様と実装に違いが生じている．日本語で一貫してソフトウェア開発が行えることは，有益であると考えている．我々は，日本語プログラミング言語“プロデル”を開発すると同時に，仕様書から実装まで日本語で一貫したプログラム開発を実践している．本研究では，比較的小規模なソフトウェア開発を例に，UML を用いて設計したものを，プロデルおよび Java でそれぞれ実装し比較した．本論文ではソフトウェアの実装を日本語で行うことによる利点や浮かび上がった問題点について報告する．

A Practice of Software Development with a Japanese Programming Language

BAMBA Yuto[†] and KAKEHI Katsuhiko[††]

We propose a practice of programming with a Japanese programming language (JPL). Japanese programmers design specifications using Japanese words in designing phase; nevertheless, they write programs using alphabet characters in implement phase. We believe in good for Japanese programmers to read and write programs using Japanese words. We have been developing a JPL “Produire”. We are trying to find advantages and research problems through software implementation with Produire. This presentation reports a case study on programming with a JPL through a small software development.

1. はじめに

現在，日本におけるソフトウェア開発では母国語である日本語で仕様書を作成し，Java に代表されるような欧米発想のプログラミング言語を使って実装することが行われている．ソフトウェア開発において，要求分析段階では，日本語表記で機能名やデータ構造を定義する．詳細設計や実装段階では，識別子(クラス名，関数名，変数名) を仕様書の日本語表記から英単語表記へ置き換える．この置き換えを行うことで仕様書に書かれた言葉(日本語表記) と，実装コードの識別子(英単語表記) との間にギャップが起こる．実装段階においても仕様書にある表現をそのままプログラムを実装することができれば，このようなギャップを解消することができると考えられる．

筆者らは，日本語プログラミング言語“プロデル [1)]”を開発中である．プロデルは，オブジェクト指向プログラミングに対応したスクリプト型の日本語プログラミング言語である．またプロデルは，実用規模のソフトウェア開発を対象とした日本語プログラミング環境を目指している．プログラミング入門や情報教育を対象とした比較的短いプログラムを日本語で書くことを目指しているものではない．既存の日本語プログラミング言語の多くは，自然な日本語の表記および語順に近い文法でプログラム文を書けることを目指しているが，プログラム構文レベルの工夫に止まっている．我々は，構文レベルの工夫では日本語プログラミングの利点は少ないと考えており，ライブラリや開発環境といったプログラム環境全体で，日本語らしい日本語プログラムの開発環境を整備し，その上で利点を議論していく必要があると考えている．そのために我々は，実用的なプログラムを開発することができる日本語プログラミング言語を設計・開発して，実際に実用規模のソフトウェアを開発する試みを行っている．

本研究では，比較的小規模なソフトウェア開発を例に，モデル駆動開発に沿って設計段階では UML で日本語を用いてモデルを作成し，実装の段階ではプロデルを用いてソフトウェアを開発した．また，これまでの Java を使ったソフトウェア開発工程と比較した．本論文では，日本語で一貫したソフトウェア開発を実践し，ソフトウェア開発の実装段階で，日本語プログラミング言語を用いることで解消するギャップやプログラム表記への対処について報告する．

2 章では，日本語で一貫したプログラミングを行うことの利点について述べる．3 章では，プロデルの特徴と言語仕様について説明し，4 章で，我々が実践したソフトウェア開発例について説明する．5 章では，日本語でプログラムを実装することを通じて定めた命名規則や言語仕様について述べ，6 章でソースコードの比較から見つか

† 早稲田大学 基幹理工学研究科
Graduate School of Fundamental Science and Engineering, Waseda University.
†† 早稲田大学 理工学術院
Faculty of Science and Engineering, Waseda University.

ったギャップについて報告する．7 章で結論を述べ，8 章でまとめを述べる．

2. 日本語プログラミング言語による日本語一貫プログラミング

2.1 日本語一貫プログラミングの利点

日本語で一貫したソフトウェア開発を行うことの利点として，ソフトウェア設計者とプログラム作成者との意思疎通がよくなるという点がある．日本語でプログラムを書くことで，プログラム作成者は日本語で書かれた設計書の言葉をそのままプログラム上で使用することができる．それにより，仕様書の表現と実装段階の表現とのギャップを埋められる可能性がある．ソフトウェア設計者も，プログラムからその構造やロジックを直感的に理解でき，デバッグや保守に利点がある．我々は，この点が有益であるかどうかを検証したいと考えており，そのためには日本語で一貫してソフトウェア開発を行うことができる環境が必要であると考えている．そこで日本語プログラミング言語を積極的に活用した日本語で一貫したソフトウェア開発を実践し，プログラミング言語の立場から利点を模索している．本研究は，比較的小規模なソフトウェア開発の実践を通じて次の点を明らかにすることが目的である．

求められる言語仕様と命名規則　日本語プログラミング言語を使って実用的なソフトウェア開発を行うために必要な言語仕様や命名規則を検討し定める．プロデルでは，オブジェクト指向プログラミングが可能であり，ソフトウェア開発に必要な最低限の開発環境は用意されている．プログラミング言語設計の立場で，実用的なソフトウェア開発に必要な言語仕様を模索する．

設計と実装の間で解消されるギャップ　ソフトウェア開発において実装段階に日本語プログラミング言語を使うことで，どのようなギャップが解消され，どのようなギャップが残されるのかを明らかにする．

2.2 日本語プログラミング言語

日本語プログラミング言語は，変数や関数を日本語で表記し，また日本語の語順に近い文法でプログラムを書くプログラミング言語である．ごく自然な日本語らしい文法でプログラムを書けることで，仕様書で用いた表現をそのままプログラムに用いることができ，その結果，仕様書と実装のギャップを少しでも埋めることができる．

一方，Java や C#のように最近のプログラミング言語の処理系は文字コードが Unicode であり，クラスや変数を日本語で名付けることができる．例えば，図 1 に示すプログラムは，Java のプログラムであり，図 2 は，図 1 の単語を日本語に置き換えた Java プログラムである．クラスやメソッド，変数だけを日本語で名付けても，プログラムの可読性に欠ける．これは，予約語(public, void)や API 群(String)は英単語が使われており，英単語表記と日本語表記が混在したプログラムとなっているからである．また，図 2 に示したような単語レベルの置き換えでは，結局は，何が動作であり何が動作対象であるかを記号や書き順だけで判別するしかない．我々は，動作の対象や付随する情報をごく自然な日本語らしい文法で書けるようにすることで，プログラムを読み書きしやすくしていることが日本語プログラミングの特徴であると考えている．また，日本語らしく書ける特徴を発揮するためには，構文のみならずライブラリなどの開発環境全体で議論する必要があると考えている．

```
public void greet(String name) {
    System.out.print(name + "さん、こんにちは");
}
```

図 1　Java プログラム

```
public void 挨拶する(String 名前) {
    システム.出力.表示する(名前 + "さん、こんにちは");
}
```

図 2　図 1 の単語を日本語に置き換えた Java プログラム

2.3 関連研究

日本語を使ってソフトウェア開発を行うことは，以前から行われている．中川らの研究 2)では，いくつかのソフトウェアを仕様書からプログラム作成まで日本語で一貫して開発している．C 言語を使って変数や関数に日本語名を用いて開発を行っており，図 2 に示した方法に近い．そのためプログラミング言語を含めた検討は，行われていない．また現在は，UML を使ったモデル駆動開発が確立しており，オブジェクト指向プログラミング言語が主流となっている．我々は，モデル駆動開発に沿って，現在のプログラミング言語環境を使って日本語で一貫したソフトウェア開発を実践したいと考えている．

畠山らは，オブジェクト指向一貫記述言語 OOJ[3)]を提案している．理工系専門分野のドメインユーザ向けに日本語で一貫したプログラム開発環境を用意している．これは，段階的に仕様を詳細化し，最終的に C++や Java で書かれたプログラムを自動生成するものである．本研究では，あくまでもソフトウェア開発の知識を持つ技術者を対象としており，実装に日本語プログラミング言語を使うことの他は，モデル駆動開発といったすでに広く行われているソフトウェア開発の方法を用いる．プログラミング言語の立場から，ソフトウェアの実装に日本語プログラミング言語を使うことの利点や課題を模索している．

日本語プログラミング言語は，以前から存在する．実用的なソフトウェア開発を目

的とした日本語プログラミング言語には，“まほろば[4]”，“Mind[5]”が挙げられる．Mindは，市販ソフトウェアの開発に利用された事例が報告されている．また，フリーソフトでは，“なでしこ[9]”や“TTSneo[10]”が挙げられる．なでしこは，豊富なライブラリを持ち，事務処理などの実用的なソフトウェア開発に利用されている．これらは，オブジェクト指向プログラミングには対応しておらず，本研究で実践しようとしているソフトウェア開発が行いづらい．

さらに日本語表記の言語仕様を採用しているプログラミング言語として“ことだまon Squeak[6]”，“ドリトル[7]”および“PEN[8]”が挙げられる．ドリトルはプロトタイプ型のオブジェクト指向プログラミング言語である．これらは情報教育で利用することを目的としており，実用的なソフトウェア開発を行うことを想定して設計されていない．

我々は，オブジェクト指向プログラミングができ，実用的なソフトウェア開発を目的とした日本語プログラミング言語を開発し，その特長を生かした利点について議論したいと考えている．本論文では，日本語プログラミング言語を用いて，実際のソフトウェア開発を行うことで，日本語プログラミング言語を行うことによる利点や課題を報告する．

3. プロデルの言語仕様

プロデルは，オブジェクト指向プログラミングに対応した日本語プログラミング言語である．プロデルでは，実用規模のプログラム開発が行えるだけのライブラリを整備しようとしている．またソフトウェア開発を実践していく過程で，言語仕様やライブラリ設計の改良を行っている．

プロデルの特徴は，次の通りである．

自然な日本語で書くプログラム　日本語として違和感のない文法でプログラムが書ける言語仕様としている．具体的には，スペースや句読点による分かち書きが必要ないことや，引数を助詞と対応させることで引数を決まった順に書く必要がないことなどがある．この点については，後ほど詳しく述べる．

オブジェクト指向プログラミング　言語仕様としてオブジェクト指向プログラミングに対応している．実用規模のソフトウェア開発として現在主流のオブジェクト指向プログラミングができることで，デザインパターンといった既存の概念を応用でき，また，ライブラリを系統的に整理することができるようにしている．

3.1 自然な日本語で書くプログラム

変数名や関数名などを日本語で書けるだけでなく，助詞を使って日本語の語順でプログラムを書くことができることが日本語プログラミング言語の特徴である．プロデルでも，ごく自然な日本語の表記や語順でプログラムを書くことで，プログラムが読みやすくなるような言語仕様としている．

助詞を使った日本語プログラムについて例を挙げて説明する．図 3 は，C 言語風に書いたメソッド呼出しの例であり，図 4 は，図 3 に相当する例をプロデルで書いたプログラムである．

```
copy(“文章.txt”, backupPath);
```

図 3　C 言語風のプログラム

```
「文章.txt」をバックアップ先へコピーする。
```

図 4　プロデルのプログラム

図 3 で示したように，一般的にメソッド呼出しは，メソッド名および実引数(必要な場合) を書く．図 4 で示したようにプロデルのメソッド呼出しも同様にメソッド名と実引数を書く．C 言語風のプログラムと異なる点は，実引数と仮引数を結びつけるために“助詞”を書く点である．実引数の直後に添えられた助詞によって，実引数と仮引数が対応付けられる．この対応付けによって，メソッド呼出しで，引数をメソッド宣言で定義された順番に書かなくても,メソッドへ正しく引数を渡すことができる．

助詞を使った文法の利便性は，名前付き引数(キーワード引数とも呼ばれる)の利便性と一致する．例えば Ada ではキーワード引数を使うことで，仮引数の名前と実引数とを対応付ける機能がある．この機能により，引数の順番を意識することなくプログラムを書くことができ,またプログラムが読みやすくなり保守にも貢献している 11). また Scala[13]や C# 4.0 以降などにも名前付き引数の機能が実装されている 12).

メソッド呼出しで引数を助詞と対応付けて書くことで，操作対象や入力値などの動作の対象や関係を明確にすることができる．またプログラムを書く際にその関数の仮引数の順番を覚えておく必要がなくなり，プログラムを読む際も，直感的に引数の関係が理解できる．

このようにプロデルは，日本語らしくメソッド呼出しを書けるようにすることで，プログラムを読み書きしやすくしている．

3.2 オブジェクト指向プログラミング

プロデルにおけるオブジェクト指向プログラミングについて，プログラム例を挙げて説明する．図 5 は，Java でクラスやメソッドを定義するプログラム例と，それに相当するプログラムをプロデルで書いたものである．なお，以下プロデルのプログラムでは，メソッド名を**太字**で表し，キーワードを下線で表す．

プロデルではクラスを宣言するために“～とは”という構文を使って書く．クラスに属するメソッドやフィールドは，“～とは”と“終わり”の間に書いて宣言する．図

5 に示したプロデルのプログラムでは，“人”というクラスを宣言している．また人クラスには，“名前”というフィールドと“名付ける”メソッド，“挨拶する”メソッドが定義されている．

メソッドは，“～手順”という構文を使って宣言する．手順とは，Java のメソッドに相当する．“～手順”から“終わり”までの間に，その手順で実行するプログラムを書く．手順の仮引数は，変数を【　】で囲むことで定義する．また，その仮引数を受ける助詞を仮引数の直後に書く．なお，図 5 に示した手順には，“【自分】”と書かれている部分がある．メソッド呼出しにおいて，“【自分】”の直後にある助詞によって対応付けられる実引数が，呼び出す手順のメッセージレシーバであることを宣言している．なお【　】という記号は，局所変数を宣言する際にも使用する．

図 5 では，“名前”フィールドの前に“-”という記号が書かれている．これは，アクセス範囲を示す記号であり，UML のクラス図と同じ表記を採用している．“+”が public を表し，“-”が private を表す．またプロデルではフィールド，変数および仮引数に対して明示的に型を宣言する必要がない．明示的に宣言する場合は，変数名に続いて“：”と型を書く．

このようにプロデルでは，オブジェクト指向プログラミングを行うための基本的な構文が用意されている．

なお，Java では主プログラムを main メソッドに書くが，プロデルでは主プログラムをソースファイルの先頭に書く．またプロデルではクラス名と同名のインスタンス変数が自動的に生成される．Java において唯一単一のインスタンスを作るために，Singleton パターンを利用する．一方，プロデルでは同名のインスタンス変数を自動生成する機能があるから，特にプログラムを書くことなく，唯一単一のインスタンスを作ることができる．

```
//Java
public class Person {
   private static string name;

   public static void setName(string value) {
      this.name = value;
   }

   public static void greet() {
      System.out.println("こんにちは"+name+"です");
   }

   public static void main(String[] args) {
      Person.setName("太郎");
      Person.greet();
   }
}
```

```
//プロデル
人を「太郎」と名付ける
人が挨拶する

人とは
    -名前：文字列

    +【自分】を、【値】と、名付ける手順
        名前は、値
    終わり

    +【自分】が、挨拶する手順
        「こんにちは」&名前&「です」を表示する
    終わり
終わり
```

図 5 Java で書かれたプログラムとプロデルで書かれたプログラム

4. 日本語一貫プログラミングの実践

4.1 航空便座席予約システム

本章では，航空便座席予約システムの開発を例に，仕様書作成から実装まで日本語で一貫してソフトウェア開発を行った過程について述べる．

航空便座席予約システムは，早稲田大学情報理工学科におけるソフトウェア工学の講義で使われた教材である．講義資料には，UML で書かれた仕様書と Java で書かれたソースコードが用意されている 15)．概念設計から要求分析まで日本語で仕様書を作成し，Java に合わせて，基本設計および詳細設計を英単語の組み合わせの名前に置き換え，実装する流れで資料が用意されている．

4.2 仕様

仕様書は UML2.0 に従って書かれた UML 図である．日本語で一貫して書くにあたり新たに基本設計および詳細設計を，クラス図，シーケンス図および状態遷移図を使って日本語で作成した．図 6 は，航空便座席予約システムにおける“座席検索”と“座席予約”のシーケンス図である．

4.3 実装

航空便座席予約システムの仕様書をもとにプロデルで動作するプログラムを実装した(ケース A)．また比較のため同じ要求仕様を使って航空便座席予約システムを Java で実装した(ケース B)．なお，3 章で述べたプログラミング言語仕様の違いによるものを除いてプログラムのロジックは同じである．

ケース A プロデルで実装　概念設計から詳細設計まで日本語で一貫して仕様書を作成し，プロデルで実装した．

ケース B　Java 上で識別子は日本語で実装　概念設計から詳細設計まで日本語で一貫して仕様書を作成した上で，Java 上で識別子に日本語を用いて実装した．なおデータ構造に関するクラスなど Java で提供される標準ライブラリは，日本語に置き換えずにそのまま利用した．

5. 日本語プログラミングによる実装

本章では，航空便座席予約システムをプロデルで実装したときに生じた仕様書の表現とプログラムの表現の違いや Java で実装したときとの表記の違いによって生じた問題点を示し，それを受けて対処策を紹介していく．

Vol.2011-SE-171 No.17
2011/3/15

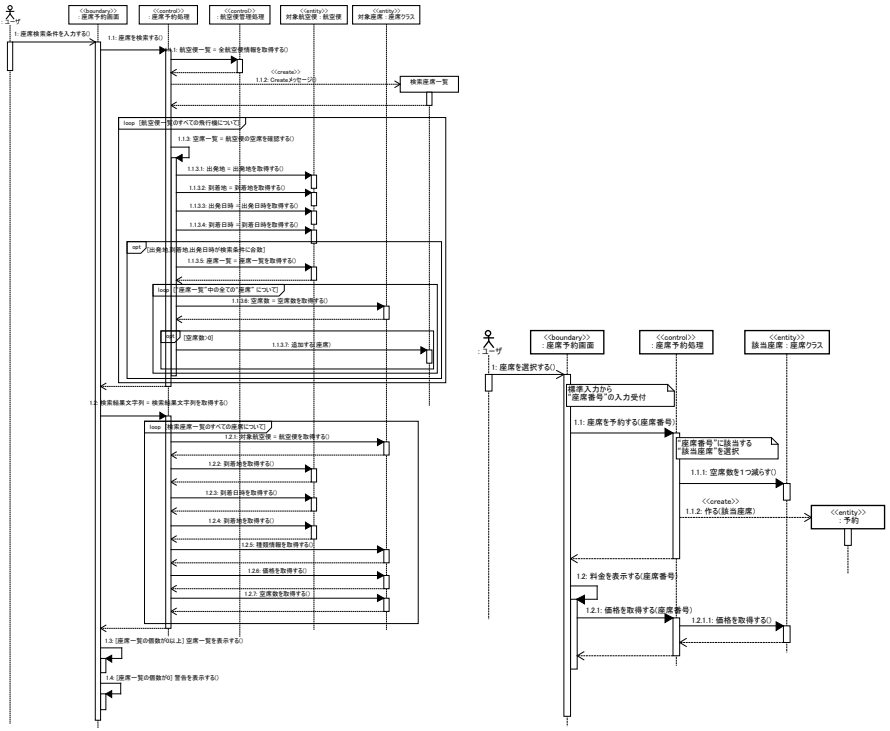

図 6 座席検索シーケンス図および座席予約シーケンス図

5.1 英単語表記と日本語表記

英単語と日本語では，表記による書き分け方が異なる．英単語を組み合わせてプログラム中の名前とする場合は，大文字・小文字の使い分けでその表す対象を区別することが行われている．例えば，Java の仕様書では，クラス名に Pascal 表記を使い，インスタンス名やメソッド名に Camel 表記を使うように推奨している．Java API 群では，実際にこの推奨規則に則った命名が行われている 14)．日本語の名前を使う場合は，識別対象を書き分けるのに，大文字・小文字の書き分けが使えないから，これとは別の工夫をするほかはない．本章では，日本語の名前を使う場合に，命名対象をうまく区別するために採った書き分け方の工夫を紹介する．

5.2 クラス名とインスタンス名

自然言語では，クラスとインスタンスをともに同じ名前で呼ぶことが多い．一方，プログラムの上ではこれらを区別したいので，実質的に同じ名前となることがある．そのために，それらを書き分ける工夫が必要となる．日本語で書き分けるのに，すぐに思いつくのは，同じ名前を漢字で書くか，かなで書くかで区別する方法である．ところが，かな綴りの名前は，日本語プログラミング言語でのプログラムの中では，構文を表すための部分と区別がつきにくく読み取りにくい．名前としては，漢字列やカタカナ列を使うのが視認しやすい．

そこで，名前には漢字列やカタカナ列を原則として当てることにした，その上で，クラス名には簡潔なものを選び，インスタンス名には個を示すものをクラス名に前置してできる名前を用いることにした．表 1 は，航空便座席予約システムをプロデルで実装したときのインスタンスの名付け例である．

表 1 インスタンス名の例

クラス	インスタンス名
空港	出発空港，到着空港,該当空港
日時	出発日時，予約日時，検索出発日時
座席	エコノミー座席，予約座席，該当座席

5.3 配列の名前

配列の名前も，そこに並ぶデータの型（クラス）と密接な名前としたいことが多い．名前に英単語を使っている場合には，単数・複数の書き分けを準用する方法がよく用いられる．表 2 は，航空便座席予約システムを Java で実装したときの配列の変数名の名付け例である．

表 2 配列の変数名(Java)

配列の型	変数名
Airplane	airplanes
Seat	seats

日本語の名前を使う場合，これにならって“航空機”クラスの配列に“航空機たち”とか“航空機ら”とする方法が考えられる．ただ，こうした接尾詞をつけた名前は普段使わないだけになんとなく落ち着かない．一般に，こうした配列は，何らかの条件を満たすものを列挙するのに使うことが多いことから，“一覧”を後置する方法を優先して採用した．表 3 は，航空便座席予約システムをプロデルで実装したときの配列の変数名の名付け例である．

表 3 プロデルでの配列変数名の例

配列の型	変数名
航空機	航空機一覧
座席	座席一覧

5.4 実装時に導入されるものの名前

クラスやメソッド, フィールドの名前は, 設計段階で決まっている. これに対して, カウンタ, イテレータ, バッファなどは, 詳細設計や実装段階になって, はじめて決める. 実装段階では, さらにメソッドの局所変数や各種作業の一時変数も導入される.

与えられた全候補それぞれについて作業を行う反復処理に使われるイテレータ変数については, その候補となるクラス名の前に“対象”を添えた名前をつけることにした. 表 4 は, プロデルでイテレータ変数を使用したときの名付け例である.

表 4 プロデルでのイテレータ変数名の例

配列の型	イテレータ変数名
航空機一覧	対象航空機
座席一覧	対象座席

メソッドの仮引数と, その仮引数の値に更新するフィールドとは, 関連付けがわかる名前を与えておきたい. しかし, わざわざ名前を考え出すのも面倒である. そこで, 仮引数名を更新対象のフィールド名の省略形にすることが多い. 表 5 は, Java で英単語を用いてフィールドを名付けた例である. 日本語の名前にも, 省略形を用いる方法は適用可能であった. プロデルでは, 表 6 に示したような省略形を用いて名付けた.

表 5 省略形の仮引数(Java)

フィールド名	仮引数名
departingAirport	deptAirport
arrivingDate	arrDate

表 6 省略形の仮引数(プロデル)

フィールド名	仮引数名
出発空港	発港
到着日時	着日時

5.5 みなす文

プロデルを用いてプログラムを実装する時に, 複数の引数を持つメソッドについて日本語らしい表記で書くことができなかった. 例えば, 図 7 で示すような Java プログラムを考える. ここで示した showInformation メソッドは 3 つの引数を持つ. 図 8 は, 図 7 で示したプログラムをプロデルで書いたものである. プロデルでは, メソッドの引数の直後に助詞を付けなければならない. 図 7 のように 3 つ以上の引数を持つメソッドを定義する場合に, 日本語として自然で適切な助詞を付けることが難しくなることが多い. そこで, Java では 3 つの引数を渡したのに対して, プロデルでは 1 つの配列にまとめて渡すことにする. そして, “みなす” 文を導入し, 受け取った引数をみなす文によって局所変数に展開することで工夫した. なお, みなす文は, Ruby における多重代入に相当する操作である.

このように, 違和感のない日本語で書くことができるように適宜, プロデルの言語仕様についても改良を加え, 日本語としてできる限り違和感のない表現でメソッド呼出し式を書くことができるようにしている.

```
//Java
showInformation("H10D8100", "山田太郎", 2010);

public static void showInformation(String id, String name, String year) {
    System.out.println(name + "(" + id + ") " + year);
}
```

図 7 Java で複数の引数を持つメソッド

```
//プロデル
｛「H10D8100」, 「山田太郎」, 2010｝について情報を表示する

【データ】について、情報を、表示する手順
        データを｛学籍番号, 名前, 入学年度｝とみなす
        名前&「(」&学籍番号&「) 」&入学年度を表示する
終わり
```

図 8 プロデルのみなす文

5.6 例外処理

航空便座席予約システムの実装を通じて, 仕様書の表現をそのままプログラムに書くだけでは, 不足している点があった. 例外処理のようなプログラミングの都合上, 必要な表記は, 仕様書に書かれていない.

例えば, 図 9 の“**座席を予約する**”手順には, 例外処理が書かれている. “例外監

視”の直後から“発生した場合”の直前までの間に書かれたプログラムで例外が発生した場合，“発生した場合”直後から“監視終わり”直前のプログラムが実行される．これは，Java の try-catch 文に相当する．この例外処理は，ユーザから不正な入力が与えられた際に，プログラムが予期せず終了してしまうことを回避するものである．

このように仕様書に書かれたソフトウェアそのものの動作の記述と，計算機やプログラミングの都合上，必要な記述とが混在していることが多かった．仕様書からプログラムとして実装する段階では，日本語表記と英単語表記との違いによって生まれるギャップだけでなく，ユーザドメイン固有の流れと，計算機の処理の流れとにも，ギャップが生じていることが明らかになった．

6. ソースコードの比較評価

航空便座席予約システムを実装した 3 つのソースコード(ケース A，ケース B および講義資料にある Java で英単語を用いた実装) を比較する．表 3 は，ソースコードの行数，文字数および識別子数を示したものである．文字数には，スペース文字やタブ文字を含まない．識別子数とは，識別子として使用された異なる単語の数を表し，別目的に重複して使われた名前も 1 つとして数えた．

表 7 を見るとおり，ソースコードの行数や識別子数に関して見ると，日本語表記と英単語表記との間に大きな違いはない．プロデルで実装した場合に，行数や識別子数が少ないのは，3 章で述べたプロデルと Java の言語仕様との違いによるものである．また，ソースコードの文字数に関しては日本語表記の方が少ない．この理由は，多くの場合英単語よりも日本語単語の方が短い文字数で表すことができるからである．

日本語表記の識別子を用いてもケース A とケース B とで差が生じた理由としては，ケース B のソースコードで Java の予約語やライブラリが英単語表記であったことが大きいと考えられる．次にそれぞれの場合について比較する．

表 7 ソースコードの比較

実装言語	行数	文字数	識別子数
プロデル	359	5,855	**89**
Java (日本語)	372	8,754	**98**
Java (英単語)	370	11,330	**95**

6.1 プロデルで開発した実装と日本語を用いた Java での実装との比較

図 9 は，図 6 をもとに航空便座席予約システムの座席検索および座席予約を行うプログラムをプロデルで書いたものの一部である．図 10 は，図 6 をもとに航空便座席予約システムの座席検索および座席予約を行うプログラムを Java で書いたものの一部である．

図 10 で示すように, 今回 ArrayList クラスなどの Java にある標準クラスライブラリを，そのまま利用した．Java で日本語を用いた場合であっても，Java のキーワード，データ型，例外クラスおよび Java の標準クラスのメソッドや属性は，英単語表記のままである．単に，仕様書に書かれた日本語を実装に用いるだけでは，英単語表記と日本語表記が混在したプログラムとなる．

一方，図 9 で示すようにプロデルでは，Java における ArrayList に相当する標準クラスが用意されている．プロデルの標準ライブラリはクラスやメソッドが日本語で名付けられている．そのことから図 9 に英単語を組み合わせた表現は，見当たらない．

```
座席予約処理とは
    -検索座席一覧：配列

    +【自分】から、【検索条件】で、座席を、検索する手順
        【出発地】。【到着地】。【出発日】
        検索条件を〔出発地，到着地，出発日〕とみなす

        【航空便一覧】は、航空便管理処理の全航空便
        検索座席一覧は、〔〕
        航空便一覧のすべての対象航空便について、それぞれ繰り返す
            検索条件で対象航空便から空席を確認する
        繰り返し終わり
    終わり

    +【自分】から、検索結果文字列を、取得する手順
        【航空便情報一覧】は、〔〕
        検索座席一覧のすべての座席について、それぞれ繰り返す
            【対象航空便】は、座席の航空便
            航空便情報は、「［対象航空便の出発空港の地名］,」&
                    「［対象航空便の到着空港の地名］,」&
                        「［対象航空便の出発日時］,」&
                            「［対象航空便の到着日時］,」&
                            「［座席の種類情報］,」&
                            「［座席の値段］,［座席の空席数］

            航空便情報一覧へ航空便情報を加える
        繰り返し終わり
        航空便情報一覧を返す
    終わり
```

```
    +【自分】で、【座席番号】へ、座席を、予約する手順
        【該当座席】
        例外監視
            該当座席は、検索座席一覧（座席番号）
            該当座席から空席数を減らす
            予約 (該当座席) を作る
            （座席番号から値段を取得したもの）を返す
        発生した場合
            0を返す
        監視終わり
    終わり

    +【自分】から、【座席番号】で、値段を、取得する手順
        【該当座席】
        該当座席は、検索座席一覧（座席番号）
        該当座席の値段を返す
    終わり

    -【検索条件】で、【対象航空便】から、空席を、確認する手順
        【出発地】。【到着地】。【出発日時】
        検索条件を〔出発地，到着地，出発日時〕とみなす
        【検索出発日時】は、出発日時を日時形式化したもの
        【対象出発日時】は、対象航空便の出発日時

        もし出発地が対象航空便の出発空港の地名かつ
                到着地が対象航空便の到着空港の地名かつ
                検索出発日時の年が対象出発日時の年かつ
                検索出発日時の月が対象出発日時の月かつ
                検索出発日時の日付が対象出発日時の日付なら

            【座席一覧】は、対象航空便の座席一覧
            座席一覧のすべての座席について、それぞれ繰り返す
                もし座席の空席数が0以上なら
                    検索座席一覧へ座席を加える
                もし終わり
            繰り返し終わり
        もし終わり
    終わり
終わり
```

図 9 プロデルで実装した航空便座席予約システム プログラム(一部)

```
public class ReservationControl {

   private ManagementControl managementControl =
      ManagementControl.getInstance();
   private ArrayList<SeatClass> foundSeatClassList;

   public void findSeats(String departingLocation,
         String arrivingLocation, String departingDate) {
      Airplane[] allAirplanes = managementControl.getAllAirplanes();
      foundSeatClassList = new ArrayList<SeatClass>();
      for (Airplane plane : allAirplanes) {
         Date queryDepDate = null;
         try {
            queryDepDate = new Date(departingDate);
         } catch(IllegalArgumentException iae) {
            return;
         }
         findMatchedVacancies
                  (departingLocation, arrivingLocation, plane, queryDepDate);
      }
   }

   public String[] getResultSeatStrings() {
      ArrayList<String> foundSeatClassStringList = new ArrayList<String>();
      for (SeatClass seat : foundSeatClassList) {
         Airplane plane = seat.getAirplane();
         foundSeatClassStringList.add(
               plane.getDepartingAirport().getLocation()
               + "," + plane.getArrivingAirport().getLocation()
               + "," + plane.getDepartingDate()
               + "," + plane.getArrivingDate()
               + "," + seat.getClassInfo()
               + "," + seat.getPrice()
               + "," + seat.getVacancy());
      }
      String[] foundSeatClassStrings = new String[foundSeatClassStringList.size()];
      return (String[]) foundSeatClassStringList.toArray(foundSeatClassStrings);
   }

   public void reserveSeat(int seatNumber) {
      SeatClass seat;
      try {
         seat = (SeatClass) foundSeatClassList.get(seatNumber - 1);
         seat.decrementVacancy();
         new Reservation(seat);
      } catch(IndexOutOfBoundsException ioobe) {
      }
   }

   public int getPrice(int seatNumber) {
      SeatClass seat;
      try {
         seat = (SeatClass)foundSeatClassList.get(seatNumber - 1);
         return seat.getPrice();
      } catch(IndexOutOfBoundsException ioobe) {
         return 0;
      }
   }

   private void findMatchedVacancies(String departingLocation,
                String arrivingLocation, Airplane plane, Date queryDepDate) {
      Date targetDepDate = plane.getDepartingDate();
      if(departingLocation.equals(plane.getDepartingAirport().getLocation())
            && arrivingLocation.equals(plane.getArrivingAirport().getLocation())
            && queryDepDate.getYear() == targetDepDate.getYear()
            && queryDepDate.getMonth() == targetDepDate.getMonth()
            && queryDepDate.getDate() == targetDepDate.getDate()
      ){
         SeatClass[] seats = plane.getSeats();
         for (SeatClass seat : seats) {
            if(seat.getVacancy() > 0) {
               foundSeatClassList.add(seat);
            }
         }
      }
   }
}
```

図 10 航空便座席予約システム Java ソースコード一部

6.2 プロデルで開発した実装と Java で開発した実装との比較

一般的なソフトウェア開発として多い Java 上で英単語の組み合わせを用いて実装した場合について述べる．

仕様書にある日本語表記を実装する言語にあわせて英単語を用いると，仕様書の言葉と実装時の識別子との変換表を用意しなければならない．プログラムに改変を加える時や動作テストの時に，ソースコードからだけで仕様書との対応を探すことは難しい．そのことから，単語の対応を確認するために対応表を用意する必要がある．より大規模なソフトウェア開発を行う場合には，対応表が膨大になると考えられる．

表 8 は，航空便座席予約システムを Java で英単語を用いて実装する場合における，仕様書の日本語表記と，実装時の英単語表記との対応表の一部である．この例では 111 対の単語の対応があった．表 7 の識別子数よりも対応単語数が多いのは，英単語の省略表現で名付けている変数を，日本語名では省略せずに名付けるなどして，同じ名前でも定義する場所によって対応する日本語名を変更したからである．ケース A およびケース B では，仕様書の表現をそのまま用いることから，このような対応表は不要であった．

この比較を通じて，実装段階での仕様書の日本語表記から英単語表記への置き換えがなくなり，ギャップの一つを解消することができた．

表 8 仕様書の日本語と実装の英単語の対応表(一部)

日本語名	英単語名	日本語名	英単語名
管理処理	managementControl	座席を予約する	reserveSeat
検索座席リスト	foundSeatClassList	座席番号	seatNumber
座席を検索する	findSeats	座席	seat
出発地	departingLocation	値段を取得する	getPrice
到着地	arrivingLocation	座席番号	seatNumber
出発日	departingDate	座席	seat
航空便一覧	plane	航空便の空席を確認する	findMatchedVacancies
対象航空便	allAirplanes	出発地	departingLocation
検索出発日時	queryDepDate	到着地	arrivingLocation
空席一覧を作成する	getResultSeatStrings	出発日	plane
結果	foundSeatClassStringList	検索出発日	queryDepDate
座席	seat	対象出発日	targetDepDate
航空便	plane	座席一覧	seats
航空便情報一覧	foundSeatClassStrings	座席	seat

7. 考察

本実践を通じて，ギャップには 2 つあることを示した．一つは識別子(クラスや変数などの名前)の差であり，もう一つは設計と実装プログラムとの差である．識別子の差とは，設計上では日本語で表記していた物や動作を実装段階で英単語表記に直したことによる差である．また設計と実装プログラムとの差とは，設計段階で表現されている本来の処理の流れとは別にプログラミングの都合のために追加されるプログラムである．例外処理は，その一例である．現在のソフトウェア開発では，実装段階でこの 2 つのギャップが同時に発生している．日本語で一貫したプログラミングを行うことで，このうちの識別子の差によるギャップが解消することができる．

また本実践では，現状のプロデルで用意された言語仕様でソフトウェアを実装することができ，ほとんど言語仕様についての改良を行う必要はなかった．この点に関しては，大規模なソフトウェア開発について日本語で一貫したプログラミングを行うこ

とによって，さらに検証が必要であると考えている．

一方，日本語プログラミング言語を使った日本語で一貫したソフトウェア開発の利点を示すには，評価方法についても課題がある．プログラミング言語仕様やソースコード上の違いに限らず,デバッグや保守におけるプログラムの理解しやすさなど,様々な方法で実験し，評価を行うべきであると考えている．

8. まとめ

我々は，本研究で仕様書から実装まで日本語で一貫したソフトウェア開発を実践した．本論文では，日本語で一貫した開発を行うために日本語プログラミング言語に求められる言語仕様について挙げ，日本語で一貫した開発を行うことで明らかになったギャップについて報告した．仕様書の表現をそのままプログラムとして書くことで，仕様書と実装との名前のギャップが解消することができる．また，仕様書にあるユーザドメイン固有のソフトウェアのメインの動作だけでなく，それらをプログラミングの動作に置き換える時にもギャップがあることが明らかになった．

我々は，本例に限らず実際のソフトウェア開発において日本語で一貫したソフトウェア開発を実践し，また同時に日本語らしくプログラムを書くために日本語プログラミング言語の言語仕様やライブラリ設計を模索する必要があると考えている．

なお，プロデルは，Web に公開しており，誰でも利用可能である．

謝辞 早稲田大学情報理工学科鷲崎弘宜先生には，研究題材を提供していただき，また本研究に関するご意見を頂きました．感謝いたします．

参考文献

1) 日本語プログラミング言語「プロデル」, http://rdr.utopiat.net/, (2011 年時点).
2) 中川正樹ほか: 日本語プログラミングの実践とその効果, 情報処理学会論文誌, Vol.35, No.10 (1994).
3) 加藤木和夫, 畠山正行ほか: オブジェクト指向プログラム設計記述言語 OOPD とその記述環境, 情報処理学会論文誌, Vol.43, No.5 (2002).
4) 今城哲二ほか: 日本語プログラム言語“まほろば”の言語仕様, 情報処理学会研究報告 (SE)(2001).
5) 木村明, 片桐明: 日本語プログラミング言語 Mind について, 情報処理学会第 16 回プログラミング言語研究会, 1988, pp.25-32.
6) 杉浦学ほか：アルゴリズム構築能力育成の導入教育：実作業による概念理解に基づくアルゴリズム構築体験とその効果，情報処理学会論文誌，Vol.49, No.10, pp.3409‐3427 (2008)..
7) 中谷多哉子, 兼宗進ほか: オブジェクトストーム：オブジェクト指向言語による初中等プログラミング教育の提案, 情報処理学会論文誌, Vol.43, No.6 (2002).
8) 西田知博ほか: 初学者用プログラミング学習環境 PEN の実装と評価, 情報処理学会論文誌, Vol.48, No.8 (2007).
9) クジラ飛行机: 日本語プログラム言語「なでしこ」, http://nadesi.com/ (2011 年時点).
10) 日本語プログラミング言語「TTSneo」, http://tts.utopiat.net/, (2011 年時点).
11) Ian Sommerville, Ron Morrison: Software Development with Ada, Addison-Wesley, 1987.
12) MSDN: C# プログラミング ガイド - 名前付き引数と省略可能な引数, http://msdn.microsoft.com/ja-jp/library/dd264739.aspx, (2011 年時点).
13) The Scala Language Specification, http://www.scala-lang.org/node/198, 2011 年時点.
14) Sun Microsystems: Code Conventions for the Java Programming Language, http://www.oracle.com/technetwork/java/codeconventions-135099.html, (2010 年時点).
15) 鷲崎弘宜ほか: ソフトウェア工学講義資料, 早稲田大学情報理工学科ソフトウェア工学講義資料(2008).